$Revision: 1.7 $ $Author: death $ $Date: 93/04/04 23:30:29 $

THIS DOCUMENT IS NOT COMPLETED. IT ONLY CONTAINS AN OVERVIEW DESCRIPTION, AND NOT METHOD INFORMATION. MAYBE THAT WILL COME LATER.

RegionConverter

INHERITS FROM	ResultObject
DECLARED IN	RegionConverter.h

CLASS DESCRIPTION

The purose of this class is simple: It converts a Macintosh Region into a series of x,y coordinates suitable for construction of a path in PostScript. I think it's questionable how 'good' a class this is, in that it's function, etc, isn't tremendously Object Oriented. It's basically a bit of the PictConverter that looks like it's just big and complex enough that I wanted to get it into a separate package. However, it's basically a part of the PictConverter class, functionally.

It's first worth explaining what a region is, and then explaining some about the data structures used by this class during conversions.

A Mac region is a data structure that (as best I have been able to deduce) has this form: 2 bytes describing the length of the whole (including these bytes), 8 bytes of a bounding/clipping rectangle, and then region data. Region data is organized into 'lines'. Each line is of the form: 2 byte Y coordinate (signed int), and then 1 or more pairs of x coordinates. The line is

terminated by a pair of bytes with the value 32767 (0x7FFF). These lines are arranged in order of increassing Y value (from top to bottom, since the mac's 0,0 is in the upper left). After the last line, there is an additional 32767 pair of bytes which flag the end of the whole structure. The x value pairs describe small segments on the specified y coordinate. If you were to draw all these segments, and perform the following heuristic, you would end up with the filled region: for each x coordinate, project a 'light' from top to bottom. Whenever this 'light' beam crosses one of these segments, it's value is toggled from on to off, or from off to on. Thus, two lines, one above the other, forms a square, since the top one turns the light off, forming darkness down to the second which turns it back on. you might ask: what happens if I put a third line under those two? The answer is: nothing. The drawing algorithm actually ignores lines that don't have a matching part somewhere below them to turn the light back on. Note that you can have one line above, and two non-overlapping, for instance, below, but at different y coordinates. Using the rule above, you can see that this would produce a non-square shape. That's all. This program makes a couple assumptions (I've seen no doc on the format of regions, and have had to try to figure it out by trial and error). First, it assumes that if the region size is 10, then there is no data following the rectangle. If it is not 10, then it ignores the size, relying on the terminal flag to tell it when it is done. Second, It relies on the fact that the Y coordinates are always sorted from smaller to larger values. Thirdly, it assumes that the pairs of x coordinates have the larger value second (that is, if the line has the values Y X1 X2 32767, that X2 has a larger value than X1). We also assume that if there are multiple segments on the line (y x1 x2 x3 x4 32767), that x2 and x3 shall never be the same coordinate(I believe the Mac automatically deal with merging these touching lines into one, as it should). Finally, it assumes my drawing heuristic, above, is accurate. Its worth noting that a framed region isn't drawn like a polygon, with the pen hanging down and to the right, but is istead more like a rectangle, with the line always within the boundrary.

So, how does one go about converting this thing? This class accomplishes this in a couple steps. First, each line is read in and the x coordinate pairs (now called segments) are stored in a linked list. One linked list is constructed per y coordinate of pict data. With this data in place, a moderately complex algorithm fires up. The goal here is to produce some 'shapes'

that will be written out to form a path in PS. These shapes are all non-overlapping outlines of the original pict regions. Because a single segment can be used to turn on and off multiple shapes being constructed, on can't simply go matching segments. So, This sets up three sets of data: There are the segments in the lines of data that were read in, but havne't yet been processed (the linked lists of segments, mentioned above), there's a set of shapes that are finished, or in the process of being constructed, and there's a set of 'active' segments. Processing proceeds from top to the set of lines to the bottom of the pict region data, with only one pass. Thus, the progress could be viewed as a line passing from the top of the image to the bottom, where the line itself represents the area currently being processed. This line would intersect the various shapes being accumulated. The lines of intersections are, in fact, the segments on what I call the `active' list. For each segment removed from its linked list, we compare it with all the active segments. If there are no intersections with any active segments ,it's is considered to be the start of a new shape, and is stored as a new shape, and this new shape intersection line is stored on the active list. Otherwise, if there was an intersection, then we use the parts of it that intersected the active segments to `turn off' those portions of the active segments (and thus to add new lines to the shapes being accumulated. More concretely, this means we just found a boundrary of the shape(s)). If there are parts that weren't used to terminate parts of shapes being built, they are treated like segments on their own (potentially being added to form new shapes). this, then, graudually traces the outline around a shape, until it is closed, at which point it is then ready to be written out as a series of points which can be made into a path in PS. Note that by virtue of the algorithm, what may initially seem like multiple shapes (at the start of the pass. Consider an outline of the letterW for instance) may all be merged into one by the bottom. This can make for a bit of mess in the data structures. The full details of this conversion is, unfortunately, more than I want to write here, and so I direct you to the source code and it's comments. It will assume you've read this.

One final note: So far as I could tell. the Mac PICT routines always take adjacent and touching segments on one line, and merge them into one. That is, if you open a region on the Mac, and write a rectangle with a left coord of 10, right of 20, and another with left of 20 and right of

40, you'll find just the segment 10 to 40 in your final pict file. My algorithms assume that they do not have to do this merging of adjacent or overlapping shapes on one line. IF this assumption is wrong, then the results will not be as expected.

INSTANCE VARIABLES

This section is incomplete.

METHOD TYPES

This section is incomplete.

CLASS METHODS

None

INSTANCE METHODS

none documented at this time.

BUGS AND PROBLEMS

This document is severely incomplete!

ENHANCEMENT IDEAS

Finish this doc.

CONSTANT, DEFINED TYPES AND ERROR CODES

?????

MODIFICATION HISTORY

$Log: RegionConverter.rtf,v $Revision 1.7 93/04/04 23:30:29 deathSun Apr 4 23:30:29 PDT 1993Revision 1.6 93/01/09 21:07:23 deathSat Jan 9 21:07:22 PST 1993Revision 1.5 93/01/01 11:51:30 deathFri Jan 1 11:51:30 PST 1993Revision 1.4 92/12/31 15:34:09 deathThu Dec 31 15:34:09 PST 1992Revision 1.3 92/12/05 23:07:16 deathSat Dec 5 23:07:16 PST 1992Revision 1.2 92/12/03 18:01:42 deathThu Dec 3 18:01:41 PST 1992Revision 1.1 92/11/27 19:37:51 deathFri Nov 27 19:37:51 PST 1992